## REMARKS

Claims 1-20 are amended. No new matter is added by these amendments. Claims 1-20 are pending. By amending the claims, applicant is not conceding that the claims are unpatentable over the art cited by the Office Action and is not conceding that the claims are non-statutory under 35 U.S.C. 101, 102, 103, and 112 as the claim amendments are only for the purpose of facilitating expeditious prosecution. Applicant respectfully reserves the right to pursue the subject matter of the claims as it existed prior to any amendment and to pursue other claims, in one or more continuation and/or divisional applications. Applicant respectfully requests reconsideration and allowance of all claims in view of the amendments above and the remarks that follow.

### *Objections to the Specification*

The specification is objected to because: "In page 1, Attorney Docket number should be replaced by US Patent Application number." The specification is amended to recite the US Patent Application number.

The specification is objected to because: "EDVAC" "should be spelled out at the first appearance." The specification is amended to spell out EDVAC at the first appearance.

The specification is objected to because: "DASD" "should be spelled out at the first appearance." Applicant respectfully traverses these grounds for rejection because "DASD" is already defined the first time DASD is used, in the following sentence at page 7, lines 18-19 of applicant's specification: "The storage interface unit 112 supports the attachment of one or more direct access storage devices (DASD) 125, 126, and 127 [emphasis added]."

*Claim Objections*

Claims 9-12 are objected to for not including "computer-readable." Claims 9-12 are amended to recite "computer-readable."

*Rejections under 35 U.S.C. 103*

Claims 1-20 are rejected under 35 U.S.C. 103(a) as unpatentable over Bates (US Patent 6,587,967), hereinafter "Bates," in view of Akgul (US Patent Publication 2003/0074650 A1), hereinafter "Akgul." Applicant respectfully submits that the claims are patentable over Bates and Akgul because Bates and Akgul do not teach or suggest all elements of the claims for the reasons argued below.

Claim 1 recites: "the entry breakpoint is executed conditionally, wherein a plurality of threads execute the program, wherein the entry breakpoint is executed conditionally further comprises some of the plurality of threads encounter the entry breakpoint and other of the plurality of threads do not encounter the entry breakpoint," which is not taught or suggested by Bates and Akgul for the reasons argued below.

In contrast to claim 1, in Bates at Fig. 2, element 102 "a determination is made as to whether the monitor control point is a monitor entry point in a section of the monitored region," as described by Bates at column 7, lines 37-40. Thus, the Bates "monitor entry point" is not executed conditionally, as recited in claim 1, but instead Bates determines the type of its "monitor entry point," i.e., Bates determines whether its "monitor entry point" is an "entry point," (the "yes" leg of block 102) as described by Bates at column 7, line 38-39 or an "exit point," (the "no" leg of block 102) as described by Bates at column 8, lines 7-8.

In contrast to claim 1, Akgul at [0053] recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint

registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool."

Thus, Akgul only describes unconditional breakpoints, so Akgul does not teach or suggest "the entry breakpoint is executed conditionally, wherein a plurality of threads execute the program, wherein the entry breakpoint is executed conditionally further comprises some of the plurality of threads encounter the entry breakpoint and other of the plurality of threads do not encounter the entry breakpoint," as recited in claim 1.

Claim 1 further recites: "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," which is not taught or suggested by Bates and Akgul for the reasons argued below.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained within any region, and Akgul has no notion of a region, so Akgul does not teach or suggest: "if the first thread encounters the scoped breakpoint within the region, wherein the region contains the scoped breakpoint and wherein the scoped breakpoint is different from the entry breakpoint and the end breakpoint," as recited in claim 1.

Claim 1 further recites: "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," which is not taught or suggested by Bates and Akgul for the reasons argued below.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and

the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained within any region, and Akgul has no notion of a region, so Akgul does not teach or suggest: "if the identifier was saved in response to the first thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint within the region was encountered by the first thread," as recited in claim 1.

Claim 1 further recites: "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," which is not taught or suggested by Bates and Akgul for the reasons argued below.

In contrast to claim 1, the Bates control point 37 at Fig. 7 is outside the regions bounded by the "monitored region of two sections defined by statement number control points 05, 10 and 30, 32," as described by Bates at column 8, lines 31-34, so the control point 37 is not within the Bates sections bounded by 05 and 10 or 30 and 32, and the Bates sections bounded by 05 and 10 or 30 and 32 do not contain the Bates control point 37, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 3, element 122 "handle[s] [the] execution stop" at the Bates control point, Bates does not check whether that control point is contained within the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In further contrast to claim 1, when Bates at Fig. 5, elements 102, 104, 106, 110 and 122 processes the entry point, Bates does not check to determine whether a control point is contained within the section bounded by the Bates entry and exit points, so Bates does not teach or suggest "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

In contrast to claim 1, Akgul at [0053] merely recites: "The addresses of instructions where breakpoints are set are written into the breakpoint registers. Then, the value of program counter 904 is continuously compared with the values in the breakpoint registers. If a match is found, an internal interrupt 906 is generated which stops the execution and notifies the debugger tool," so the Akgul breakpoints are not contained

within any region, and Akgul has no notion of a region, so Akgul does not teach or suggest: "if the identifier was not saved, the first thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint within the region was encountered by the first thread that executes the instance of the program," as recited in claim 1.

Thus, Bates and Akgul do not teach or suggest all elements of claim 1. Claims 5, 9, 13, and 17 include similar elements as argued above for claim 1 and are patentable over Bates and Akgul for similar reasons as those argued above. Claims 2-4, 6-8, 10-12, 14-16, and 18-20 are dependent on claims 1, 5, 9, 13, and 17, respectively, and are patentable for the reasons argued above, plus the elements in the claims.

*Conclusion*

Applicant respectfully submits that the claims are in condition for allowance and notification to that effect is requested. The Examiner is invited to telephone Applicant's attorney (651-645-7135) to facilitate prosecution of this application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 09-0465.

Respectfully submitted,

Date: August 11, 2008

Owen J. Gamon
Reg. No. 36,143
(651) 645-7135

IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

**CERTIFICATE UNDER 37 C.F.R. 1.8**

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Mail Stop AF, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, or is being transmitted via facsimile to the U.S. Patent and Trademark Office, 571-273-8300, on: August 11, 2008.

Owen J. Gamon
Registration No. 36,143